NAS Technical Report; NAS-06-008
July 2006

A Scalability Study of SGI Clustered XFS Using HDF Based AMR Application

Subhash Saini, Dale Talcott, Herbert Yeung, George Myers, and Robert Ciotti

Terascale Systems Group
NASA Ames Research Center
Moffett Field, California 94035-1000, USA
{ssaini, dtalcott, hyeung, gmyers, ciotti}@mail.arc.nasa.gov

Abstract: Many large-scale parallel scientific and engineering applications, especially climate modeling, often run for lengthy periods and require data checkpointing periodically to save the state of the computation for a program restart. In addition, such applications need to write data to disks for post-processing, e.g., visualization. Both these scenarios involve a write-only pattern using Hierarchal Data Format (HDF) files. In this paper, we study the scalability of CXFS by HDF based Structured Adaptive Mesh Refinement (AMR) application for three different block sizes. The code used is a block-structured AMR hydrodynamics code that solves compressible, reactive hydrodynamic equations and characterizes physics and mathematical algorithms used in studying nuclear flashes on neutron stars and white dwarfs. The computational domain is divided into blocks distributed across the processors. Typically, a block contains 8 zones in each coordinate direction (x, y, and z) and a perimeter of guard cells (in this case, 4 zones deep) to hold information from the neighbors. We used three different block sizes of 8x8x8, 16x16x16, and 32x32x32. Results of parallel I/O bandwidths (checkpoint file and two plot files) are presented for all three-block sizes on a wide range of processor counts, ranging from 1 to 508 processors of the Columbia system.

Key words: parallel I/O, clustered file system (CXFS), benchmarking, performance evaluation, HDF5, adaptive mesh refinement, AMR.

1 Introduction

NASA's many large-scale parallel scientific and engineering applications often run for lengthy periods and require data checkpointing periodically to save the state of the computation for a program restart. These applications include climate modeling applications such as the Goddard Earth Observing System 4/5 Global Climate Model (GEOS-4/5 GCM) from the Global Modeling and Assimilation Office (GMAO), located at NASA Goddard Space Flight Center (GSFC) [1], and Weather Research Forecasting (WRF) codes [2]. In addition, such applications need to write data to disks for post-processing, for example, for visualization. Both these scenarios involve a write-only pattern using Hierarchical Data Format (HDF) files [3]. Each GEOS-4/5 file contains a single HDF-EOS grid, which in turn includes a number of geophysical variables. Some files contain 2-D variables on a longitude/latitude grid and some files contain 3-D variables on the same longitude/latitude grid but with an additional vertical dimension.

Recently, NASA, under its new National Leadership Computing System (NLCS) initiative chartered to provide resources to computationally intensive research projects in the national interest, has awarded a million hours of compute time on the Columbia system to Dr. Greg Holland and his team at the National Center for Atmospheric Research (NCAR), They will run the WRF code to simulate weather at high, cloud-resolving resolution to determine how moist convection impacts natural climate change, such as hurricane frequency and intensity [4]. This will provide an understanding of future hurricane impacts, leading to much improved understanding of how climate both influences, and is influenced by, human activities. The WRF code involves massive I/O of about 6.5 terabytes (TB) that can be accessed via either sequential or parallel HDF. Also, the National Science Foundation (NSF) and the Department of Defense (DoD) High Performance Computing Modernization Office (HPCMO) have adopted a benchmark version of WRF for testing future generation of petaflops-class

computing systems [5]. WRF will be run in the 2,048-processor shared-memory Columbia supercomputer environment, located at the NASA Advanced Supercomputing (NAS) facility at Ames Research Center under the Shared Capability Assets Program (SCAP), which funds the High-End Computing Columbia Project.

In both the GEOS-4/5 and WRF codes, the performance bottleneck is in the I/O. Therefore, it is important to understand the I/O characteristics of modern supercomputers such as Columbia and look for ways to improve I/O performance.

To get a better understanding of how the I/O systems of one of today's leading supercomputers perform, we undertook a study to benchmark the parallel I/O performance of NASA's Columbia supercomputer. In particular**,** we characterize the parallel I/O performance and scalability of SGI's CXFS (Clustered XFS) file system [6] on Columbia running an HDF-based Structured Adaptive Mesh Refinement (AMR) application for three different block sizes.

The rest of this paper is organized as follows. In Section 2, we present the architectural details of Columbia and its file system. In Section 3, we describe the HDF-based AMR application used in this study. In Section 4, we present and analyze the results of the benchmarking study. We conclude in Section 5 with a discussion of future work.

2 An Overview of Columbia

2.1 Columbia Architecture

Columbia consists of twenty 512-processor SGI Altix computers. Twelve of these are model 3700, and eight are model BX2. Because the experiments in this paper were conducted on BX2 systems, we'll confine our discussion to that architecture.

In the SGI 3700 BX2 system, eight Intel Itanium 2 processors are grouped together in a brick, called a C–brick, which is connected by a NUMALink4 interconnect to another C-brick. Each pair of processors shares a peak bandwidth of 3.2 gigabytes per second (GB/s). Peak bandwidth between nodes is 1.6 GB/s [7-11].

The SGI Altix is a Cache Coherent - Non-Uniform Memory Access (CC-NUMA) system. Local cache-coherency is used to maintain the cache coherency between processors on the Front Side Bus (FSB). Global cache coherency protocol is implemented by the Scalable Hub (SHUB) chip and is a refinement of the protocol used in the DASH computing system developed at Stanford University, which is directory based. The advantage of the directory-based cache–coherent protocol is that only the processors that are playing an active role in the usage of a given cache line need to be informed about an operation. This reduces the flow of information, at the cost of using about 3 percent of memory space for the directory.

The combination of compute processors, memory, SHUBs, and R-brick constitute the interconnect fabric called NUMALink. The SHUB is a proprietary Application Specific Integrated Circuit (ASIC) designed by SGI and fabricated by IBM, which handles the functions including: (a) global cache coherency protocol; (b) memory controller for the local memory on the node; (c) interface to I/O subsystem; (d) interface to the interconnection network with other nodes; and (e) globally synchronized high-resolution clock. The SGI Altix 3700 BX2 uses NUMALink4, a high-performance network with fat-tree network topology. In fat-tree network topology, the bisection bandwidth scales linearly with the number of processors.

Each Altix has globally shared memory. It is a single-system image (SSI) design, which means that a single memory address space is visible to all the computing system resources. SSI is achieved through the NUMALink memory interconnect. It is a Non-Uniform Memory Access Flexible (NUMAflex) system as scaling can be done in three dimensions, namely the number of processors, the memory capacity, and the I/O capacity. This NUMAflex architecture supports up to 2048 Intel Itanium 2 processors and four TB of memory.

At NAS, four of the BX2s are organized as a capability platform by interconnecting with three networks – (a) NUMALink4, (b) InfiniBand, and (c) 10 Gb Ethernet [13, 16, 18]. The InfiniBand and Ethernet interconnects connect to the other 16 boxes, as well [12-15].

2.2 File System on Columbia

In the past, the 20 Altix machines of Columbia accessed a shared Network File System (NFS) containing the users' home directories. Due to the relatively poor performance of NFS file systems,

each of the machines also had a local XFS-based scratch disk (/nobackup*i*, where i=1, 2, 3, …, 20), and users employ these scratch disks for their performance-sensitive I/O. However, this configuration was not conducive to efficient use of the Columbia system. For example, if users of host Columbia5 wanted to run an application on Columbia9, they had to ensure that files accessed by their application on /nobackup5 also existed on /nobackup9. In addition, the design of the NFS file system is to provide distributed access to files from multiple hosts, and its consistency semantics and caching behavior are, accordingly, designed for such access. A typical scientific-computing workload does not mesh well with the semantics of NFS, especially for concurrent writes. Therefore, in February 2006, the Columbia system was reconfigured to take advantage of SGI's Clustered XFS (CXFS) technology, which overcomes the problems associated with NFS and permits a more efficient shared file system.

With CXFS, the metadata about files is still managed by shared servers, but each host has direct access via Fibre Channel to the file data disks. In the systems under test (nodes C17-20), each host communicates with its domain's three metadata servers via gigabit Ethernet. The file system data blocks are accessed across four 4-Gb/s, Fibre Channel connections to dual, 2-Gb/s RAID controllers, each with 2.5 GB of cache, interfacing with 30 TB of disk space striped across 8 logical unit numbers (LUNs) of 8+1 RAID-3 [3]. Figure 1 shows the configuration as it will be when the controllers are upgraded to larger, 4-Gb/s models. The configuration under test was midway through an upgrade from 2 Gb/s Fibre Channel to 4 Gb/s. In particular, the disk controllers were still only 2 Gb/s.

3 Application Used

We have used an HDF5-based application to study the scalability and performance characterization of CXFS on Columbia. We describe HDF5 and the application below.

3.1 HDF5 Interface

HDF5 is an I/O library from the National Center for Supercomputing Applications (NCSA). HDF5 is a de facto standard in the scientific and engineering community including the NASA Earth Observing System project of the NASA Earth Observatory. Its data model consists of hierarchical data organization in a single file, typed multidimensional array storage, and attributes on datasets. Its features include C, C++, and Fortran interfaces; portable data format, optional compression (not in a parallel I/O mode); data reordering (chunking) and noncontiguous I/O (memory and file with hyperslabs). Within a dataset space, subsets may be selected when non-contiguous data access is required. These dataspace subsets are referred to as hyperslabs. In many aspects, these subsets are similar to MPI derived datatypes.

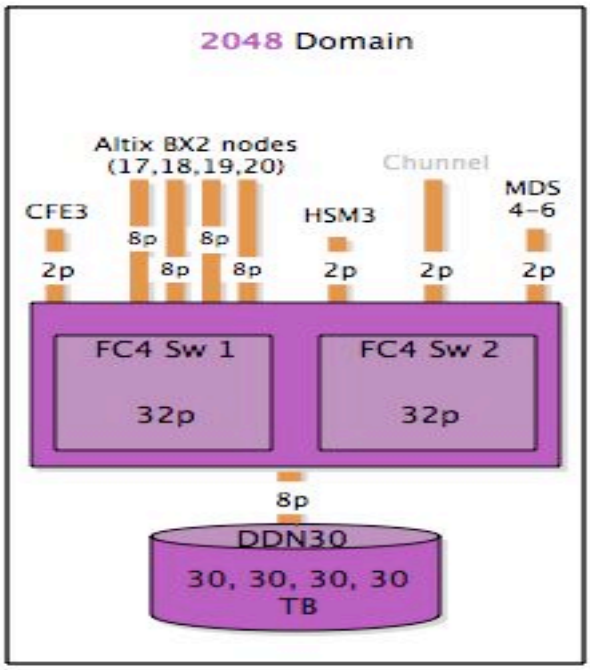

Figure 1: The Columbia 2048 cluster's CXFS I/O configuration (future).

The HDF5 files consist of groups, datasets, and attributes. Groups are like directories, holding other groups and datasets. Datasets hold arrays of typed data, where a datatype describes the type and a dataspace gives the dimensions of the array. Datatypes can be atomic (integers, floats, and others) or compound like structures of C. Attributes are small datasets associated with the file, group, or another datasets. They have a datatype and dataspace and can only be accessed as unit.

HDF5 is a bit different from previous HDF releases such HDF4. Differences include: (a) Support for files greater than 2 GB in size, even on 32-bit

platforms (as long as they conform to the LFS conventions); (b) much simpler set of objects, consisting of multidimensional arrays of data elements and grouping objects; and (c) support for threading and parallel I/O using MPI-IO.

The HDF5 API provides a property list that is an object containing properties of HDF5 files, including on-disk layout, chunking sizes, filters, a list of external HDF5 files to be "mounted" as part of the HDF5 file system within a file, and interactions with underlying I/O systems (e.g. MPI-IO). As different high-end computing systems vary a great deal in how they handle I/O, the selection of properties often depends on which computing systems is being used.

3.2 FLASH Application

Massive stars more than ten times the mass of our Sun evolve for millions of years and then die in a matter of hours in stellar explosions known as core collapse supernovae. Such supernovae are one of only two classes of supernovae in the universe. Core collapse supernovae are neutrino driven, whereas "Type Ia" supernovae occur via thermonuclear runaway and mark the death throes of less massive stars known as white dwarfs. Type Ia supernovae are the brightest thermonuclear explosions in the universe. The explosion begins when a few hot spots near the center of the white dwarf experience a runaway in their nuclear energy generation. An unstable front of turbulent combustion speeds through the star, turning most of it into iron, and blowing it apart. A first-principles understanding of these explosions eluded astrophysicists for decades. Using Columbia, researchers from the University of California, Santa Cruz, and Lawrence Berkeley National Laboratory have simulated nuclear fusion flames for enough time to see its turbulent structure develop.

The FLASH application [9] is a parallel code written in Fortran 90 using the MPI paradigm. It solves compressible, reactive hydrodynamic equations using adaptive mesh refinement (AMR) to study the problems of nuclear flashes on the surfaces of neutron stars and white dwarves, in particular X-ray bursts, type Ia supernovae, and classical novae. Algorithms used in the FLASH code are parallel adaptive mesh refinement with PARAMESH [10], compressible hydrodynamics with PROMETHEUS, a stellar EOS, and nuclear burning. PARAMESH is a package of FORTRAN 90 subroutines designed to provide an application developer with an easy route to extend an existing serial code that uses a logically Cartesian structured mesh into a parallel code with AMR. PARAMESH falls into a class of AMR techniques known as block-structured AMR. The computing paradigm that PARAMESH uses is single program multiple data (SPMD); that is, the same code executes on all the processors but the local data content modifies the program flow on each processor. The computational domain is divided into blocks of dimension 8x8x8, 16x16x16, and so on. Each block has a perimeter of four guard cells to hold the state variables of the neighboring blocks. Block hierarchy is managed by an oct-tree method, and load balancing is performed by a weighted space filling curve through the blocks, which produces a one dimensional ordering of blocks.

FLASH I/O is a smaller version of the FLASH code that simply mimics FLASH's I/O patterns. The data domain is divided into blocks distributed across the processors. The benchmark uses the parallel HDF5 library for data I/O. It produces three output files: (a) a checkpoint file, (b) a plot file for centered data, and (c) a plot file for corner data. These three output files are very different: the checkpoint file is large and dense, whereas the two plot files are smaller and sparse. The checkpoint file stores all the data variables (excluding the guard cells), the tree structure, and some additional data including the current simulation time, current time step and the number of steps. A total of 24 separate I/O operations are performed during checkpointing, one for each variable (pressure, velocity, density, energy, entropy, free energy, etc). Plot files (with and without corners) have the same format as checkpoint files but with fewer variables and half the precision (four bytes vs. eight in the checkpoint file).

4 Results

In this section we present the results of our parallel I/O experiments on performance characterization and scalability of SGI CXFS on SGI Altix BX2.

The plot in Figure 2 shows the size of checkpoint files for three block sizes – 8x8x8, 16x16x16, and 32x32x32. Here, we notice that the file with blocks

of 32x32x32 is about eight times the size the file with block size 16x16x16, which in turn is about sixteen times the size of the file with 8x8x8 blocks.

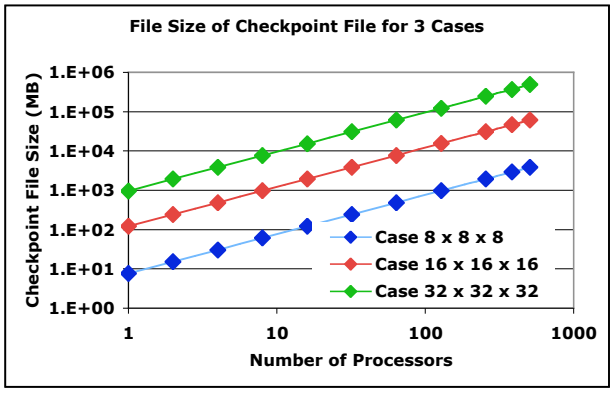

Figure 2: Size of checkpoint file for three different block sizes.

Figure 3 shows the size of a plot file with no corners for three block sizes – 8x8x8, 16x16x16, and 32x32x32. Here, we notice that the file with block size 32x32x32 is a little more than seven times the size the file blocked 16x16x16, which in turn is slightly less than seventeen times the size of the file blocked 8x8x8.

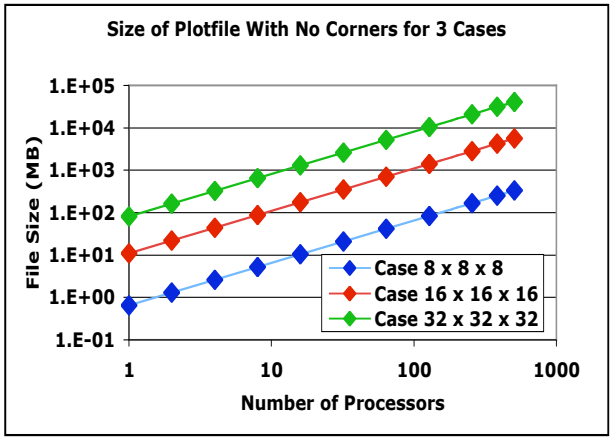

Figure 3: Size of plot file with no corners size for three different block sizes.

Figure 4 shows the size of a plot file with corners for three block sizes–8x8x8, 16x16x16 and 32x32x32. Here, the file with block size 32x32x32 is slightly less than seven times the size the file with blocks of 16x16x16, which in turn is about fourteen times the size of the file blocked 8x8x8.

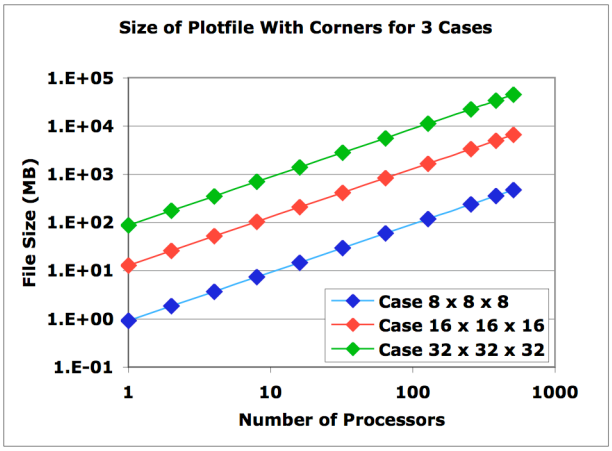

Figure 4: Size of plot file with corners size for three different block sizes.

To see the relative sizes of a checkpoint file, a plot file with no corners, and a plot file *with* corners, we have plotted these files for block size 16x16x16, shown in Figure 5. The checkpoint files are an order of magnitude larger than both plot files. Also, plot files with corners are slightly larger than the plot files with no corners.

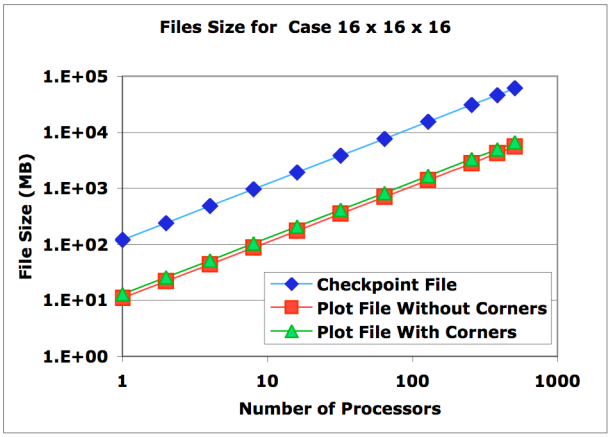

Figure 5: Size of checkpoint file and plot files with corner and no corner for a block size of 16x16x16.

Figure 6 plots the bandwidth writing checkpoint files and plot files with corners and no corners for a block size of 8x8x8 on varying number of processors. In the entire range of processors from 1 to 508, the I/O bandwidth to the checkpoint file is higher than the plot file with corners, and that in turn is higher than the plot file without corners. For all the three files, bandwidth increases from 1 processor to 16 processors—at which bandwidth is highest—and then decreases gradually. From 384 to 508 processors, they are very close to each other.

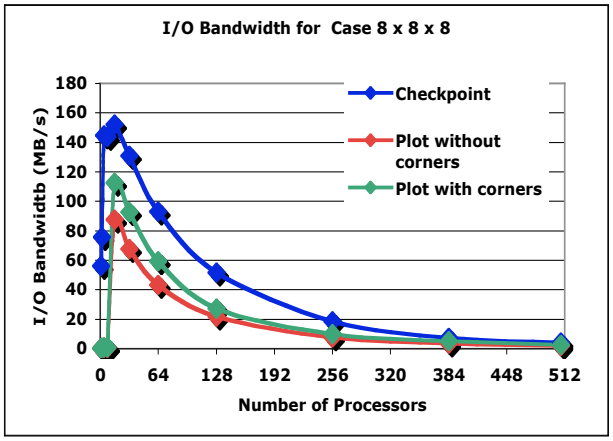

Figure 6: I/O bandwidth of checkpoint and plot files for varying number of processors for block size of 8x8x8.

The plot in Figure 7 shows bandwidth writing the checkpoint file and plot files with corners and no corners for a block size of 16x16x16 on varying number of processors. For all three files, I/O bandwidth increases from 1 processor to 32 processors, after which it starts decreasing. From 64 processors to 256 processors, bandwidth to all three files gradually decreases—checkpoint is higher than the plot file with corners, which is in turn higher than the plot file with no corners. From 256 processors to 508 processors, bandwidth to all three files continues decreasing, with bandwidth to the checkpoint file higher than bandwidth to both of the plot files, which are almost the same.

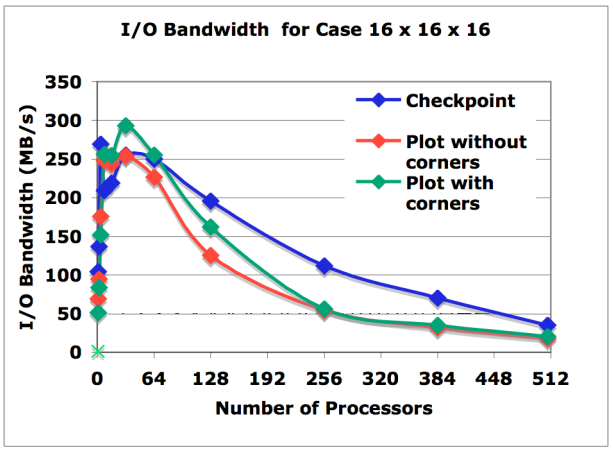

Figure 7: I/O bandwidth of checkpoint and plot files for varying number of processors for block size of 16x16x16.

Figure 8 shows the write bandwidth to the checkpoint file and plot files with corners and no corners for a block size of 32x32x32 on varying number of processors. I/O bandwidth to the checkpoint file increases from 1 to 4 processors and then starts decreasing. Bandwidth to both plot files increases from 1 processor to 8 processors. Here, results are reported only up to 128 processors because the benchmark could not run on 256, 384, and 508 processors due to a limited disk quota (400 GB) for the researchers. Trends in this figure are quite different from those of figures 6 and 7. Unlike figures 6 and 7, in the processors range from 32 to 128, bandwidth to the plot file without corners is higher than bandwidth to the checkpoint and the plot file with corners.

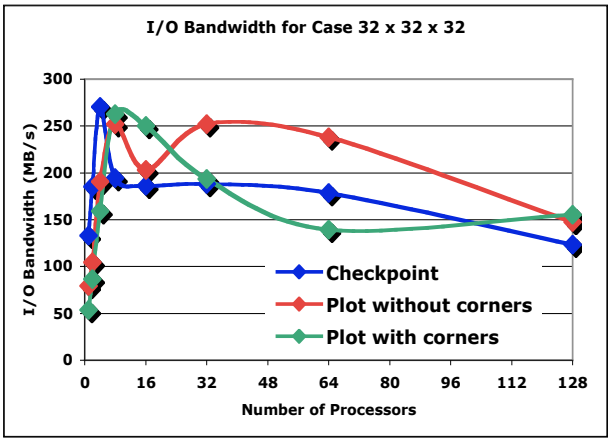

Figure 8: I/O bandwidth of checkpoint and plot files for varying number of processors on SGI Altix BX2 for block size of 32x32x32.

In Figure 9 we plot the data from figures 6 – 8 for just the checkpoint files. The I/O bandwidth for block sizes 32x32x32, 16x16x16, and 8x8x8 becomes maximum at 271 megabytes per second (MB/s) on 4 processors, 255 MB/s on 32 processors; and 152 MB/s on 16 processors, respectively. After achieving maximum, bandwidth for all three block sizes gradually decreases as the number of processors increases.

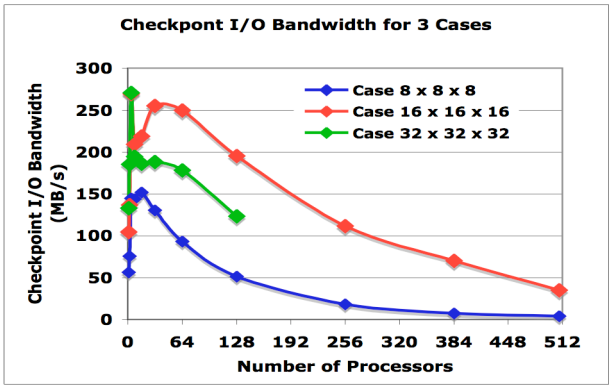

Figure 9: I/O bandwidth of checkpoint file for three block sizes on SGI Altix BX2 for various processors.

Figure 10 shows the I/O bandwidth to a plot file without corners for three block sizes of 8x8x8, 16x16x16, and 32x32x32. For both 16x16x16 and 32x32x32 the bandwidth becomes maximum at 252 MB/s on 32 processors. For block size 8x8x8 it becomes maximum at 88 MB/s on 16 processors. After achieving maximum, bandwidth for all three block sizes decreases as the number of processors increases. Where data for the 32x32x32 case are available, it has the best bandwidth except for 16 and 32 processors, where 16x16x16 is slightly better. The bandwidth for block size 16x16x16 is much higher than that of block size 8x8x8.

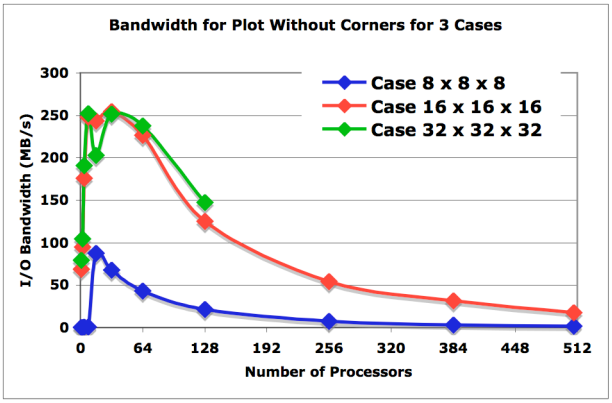

Figure 10: I/O bandwidth of plot file without corners for three block sizes on SGI Altix BX2 for various processors.

In Figure 11, the corresponding data for plot files with corners is shown. I/O bandwidth for block sizes of 8x8x8 becomes maximum at 113 MB/s on 16 processors; for block size 16x16x16 it becomes maximum at 293 MB/s on 32 processors; for block size 32x32x32 it becomes maximum at 263 MB/s on 8 processors. After achieving maximum, bandwidth for all three block sizes gradually decreases as the number of processors increases. Again, results for block size 32x32x32 are available only up to 128 processors due to limited disk space. For plot file with corners, the 16x16x16 case achieves higher bandwidth than the 32x32x32 case up to 128 processors. From all processor counts, bandwidth for block size 16x16x16 is much higher than that of block size 8x8x8.

In Figure 12 is plotted the average I/O bandwidth for three block sizes on SGI Altix BX2 for processors 1 to 128. Here average means average bandwidths of checkpoint file, plot file without corners and plot file with corners.

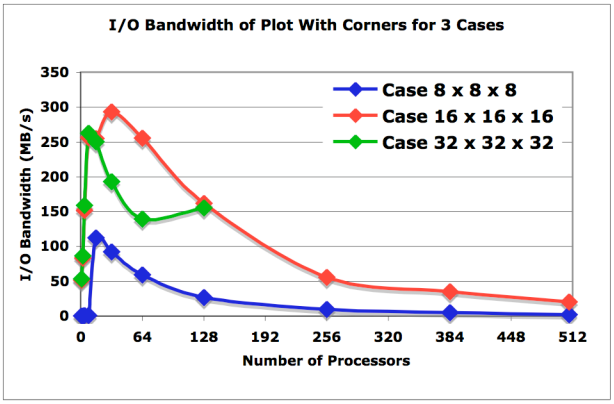

Figure 11: I/O bandwidth of plot file with corners for three block sizes for various processors.

For block size of 8x8x8, bandwidth achieves highest value of 117 MB/s at 16 processors. For block size of 16x16x16, bandwidth achieves highest value of 268 MB/s at 32 processors. For block size of 32x32x32, bandwidth achieves highest value of 237 MB/s at 8 processors. From 32 to 128 processors, average bandwidth for all the three block sizes decrease gradually. At 128 processors, average I/O bandwidth for block sizes 16x16x16 and 32x32x32 become almost same.

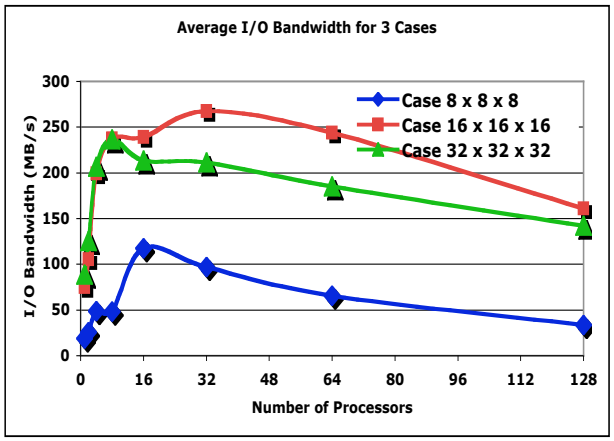

Figure 12: Average I/O bandwidth for three block sizes on SGI Altix BX2 for processors 1 to 128.

Figure 13 is same as figure 12 but extended to 508 processors. From processors 128 to 508, average I/O bandwidth for both the block sizes 8x8x8 and 16x16x16 decreases gradually, with bandwidth for block size 16x16x16 being higher. The difference decreases as the number of processors increases, and finally at 508 processors, the difference between the two becomes very small.

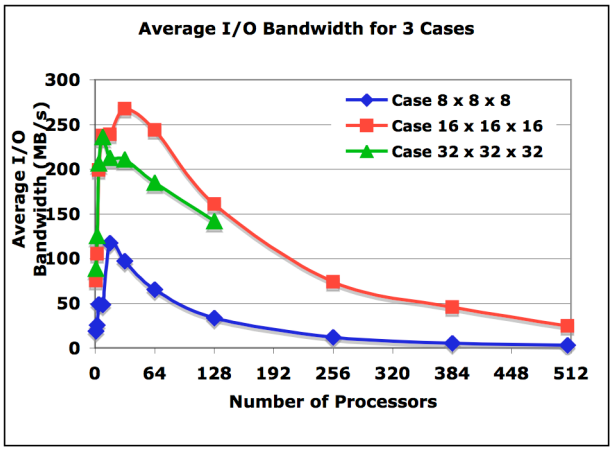

Figure 13: Average I/O bandwidth for three block sizes on the SGI Altix BX2 for processors 1 to 508.

5 Analysis and Conclusions

We ran the benchmark with a configured maximum of 500 blocks per processor. The number of blocks assigned per processor depends on the number of zones in x, y, and z direction. For a block size of 8x8x8, the number of blocks computed by the algorithm is 80. Now, the size of each record from a single processor in the 8x8x8 case is (8 bytes per variable) * (8 zones in x) * (8 zones in y) * (8 zones in z) * 80 blocks giving 327,680 bytes per variable. Multiplying by 24 variables yields 7,864,320 bytes. HDF5 adds a little overhead but the result is a checkpoint file size of about 7.6 mebibytes (2^{20} bytes) (MiB) per processor.

A similar analysis can be performed for the 16x16x16 case, except that the number of blocks per processor increases to 160 and the zones double in each dimension. The result is a record size of 2.5 MiB and a file size of about 121 MiB per processor.

For the 32x32x32 case, the number of blocks per processor remains at 160 and the number of zones doubles again. The result is about 20.6 MiB per record and 961 MiB per processor.

Examining figures 6-12, it is clear these data rates are quite poor. The theoretical bandwidth limit should be (4 data paths) * (2 Gb/s data rate), or about 1 GB/s. Other I/O benchmarks have attained values near this maximum. A large part of the problem is that Columbia's CXFS is configured with a stripe size of 1 MiB. From the preceding analysis of file sizes, we see the record size is only about 320 kibibyte (2^{10} bytes) (KiB) for the 8x8x8 checkpoint file. It is even less for the plot files. The 8x8x8 case is uniformly the slowest. The 16x16x16 and 32x32x32 record sizes are both greater than the stripe size, but there is no guarantee the I/O records are aligned to stripe boundaries. At very low processor counts, the 20 MiB 32x32x32 records seem to have an advantage over the 2.5 MiB 16x16x16 records. Very quickly though, as the processor counts increase, the more moderately sized 16x16x16 records emerge as the winner.

At larger processor counts, it is apparent the processes interfere with each other, but it is unclear how much of this interference is due to the HDF5 layer, and how much to the system I/O layer. Results of unpublished benchmarks, not using HDF5, show a less steep drop-off in output and a higher bandwidth at large process counts [19]. The function HDF5write() in HDF5 has two operations – (a) gather variables from memory and (b) scatter these variables into the file. Probably, low performance of HDF5 is due to inefficient implementation of the gather and scatter functions.

6 References

[1] Global Modeling and Assimilation Office, http://gmao.gsfc.nasa.gov/, (2006).

[2] The Weather Research and Forecasting (WRF) Model, http://www.wrf-model.org/index.php, (2006).

[3] HDF5, http://hdf.ncsa.uiuc.edu/HDF5/, (2006).

[4] The National Center for Atmospheric Research (NCAR), http://www.ncar.ucar.edu/, (2006).

[5] WRF, http://www.nsf.gov/, (2006).

[6] CXFS, http://www.sgi.com/products/storage/tech/file_systems.html.

[7] S. Saini Hot Chips and Hot Interconnects for High End Computing Systems, M4, *IEEE SC* 2004, Pittsburgh, (2004).

[8] S. Saini, Performance Comparison of Columbia 2048 and IBM Blue Gene/L, SGIUG 2005 Technical Conference and Tutorials, June 13-16, 2005 - Munich, (2005).

[9] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes,M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes, The Astrophysical Journal Supplement Series, 131:273-334, (2000).

[10] MacNeice, P., Olson, K. M., Mobarry, C., de Fainchtein, R., & Packer, C., PARAMESH: A parallel adaptive mesh refinement community toolkit,, Comput. Phys. Commun., 126, 330, (2000)

[11] S. A. Jarvis, D. P. Spooner, H. N. Lim, C. Keung, J. Cao, S. Saini, and G. R. Nudd, Performance Prediction and its Use in Parallel and Distributed Computing Systems, *Future Generation Computer Systems* special issue on System Performance Analysis and Evaluation, (in press) (2006).

[12] S. Saini, R. Ciotti, T. N. Gunney, T. E. Spelce, A. Koniges, D. Dossa, P. Adamidis, R. Rabenseifner, S. R. Tiyyagura, M. Mueller, Rod Fatoohi, Performance Evaluation of Supercomputers using HPCC and IMB Benchmarks *IPDPS 2006, PMEO*, April 25-29, Rhodes, Greece (2006).

[13] S. Saini and R. Fatoohi, and R. Ciotti, Interconnect Performance Evaluation of SGI Altix 3700 BX2 Cray X1, Cray Opteron Cluster, and Dell PowerEdge, *IPDPS 2006, PMEO*, April 25-29, Rhodes, Greece, (2006).

[14] S. Saini, R. Ciotti, T. N. Gunney, T. E. Spelce, A. Koniges, D. Dossa, P. Adamidis, R. Rabenseifner, S. R. Tiyyagura, M. Mueller, Rod Fatoohi, Performance Comparison of Cray X1 and Cray Opteron Cluster with Other Leading Platforms Using HPCC and IMB Benchmarks, *CUG 2006*, May 8-11, 2006 Lugano, Switzerland, (2006).

[15] S. Saini, P. Adamidis, R. Fatoohi,; J. Chang and R. Ciotti, Performance Analysis of Cray X1 and Cray Opteron Cluster, *CUG 2006*, May 8-11, 2006 Lugano, Switzerland, (2006).

[16] B. B. Karki, V. Yerraguntla, H. Kikuchi, and S. Saini A Parallel Molecular Dynamics Algorithm for Polycrystalline Minerals, The 2005 International MultiConference in Computer Science & Computer Engineering Las Vegas, Nevada, USA, June 27-30, 2005 *MSV* 2005: 201-207, (20050

[17] R. Biswas, M. J. Djomehri, R. Hood, H. Jin, C. Kiris, and S. Saini An Application-Based Performance Characterization of the Columbia Supercluster, , *IEEE/ACM SC 2005*: 26, (2006).

[18] S. Saini, R. Biswas, S. Gavali, H. Jin, D. C. Jespersen, M. J. Djomehri, and N. Madavan, NAS Experience with the Cray X1, *CUG 2005*, May 16-19, Albuquerque, New Mexico, USA, (2006).

[19] S. Saini and D. Talcott, Unpublished, (2006).